CLAN SU20-1

Los Alamos National Laboratory is operated by the University of California for the United States Department of Energy under contract W-7405-ENG-36

MASTER

TITLE EXISTING COMPUTER APPLICATIONS MAINTAIN OR REDESIGN: HOW TO DECIDE?

AUTHOR(S) Linda Brice, ADP-2

SUBMITTED TO December 1-4, 1981, Fairmont Hotel, New Orleans, Louisiana
Sponsored by: CMG XII International Conference on Computer Performance Evaluation Existing Computer Applications, Maintain or Redesign: How to Decide?
(will be published if accepted)

By acceptance of this article, the publisher recognizes that the U.S. Government retains a nonexclusive, royalty-free license to publish or reproduce the published form of this contribution, or to allow others to do so, for U.S. Government purposes.
The Los Alamos National Laboratory requests that the publisher identify this article as work performed under the auspices of the U.S. Department of Energy.

Los Alamos Los Alamos National Laboratory
Los Alamos, New Mexico 87545

FORM NO 836 R4
ST NO 2629 5/81

Existing Computer Applications

Maintain or Redesign: How to Decide?

ABSTRACT

Maintainence of large applications programs is an aspect of performance management that has been largely ignored by those studies that attempt to bring structure to the software production environment. Maintenance in this paper means: fixing "bugs", modifying current design features, adding enhancements, and porting applications to other computer systems. It is often difficult to decide whether to maintain or redesign. One reason for the difficulty is that good models and methods do not exist for differentiating between those programs that should be maintained and those that should be redesigned.

This enigma is illustrated by the description of a large application case study. The application was monitored for maintenance effort, thereby providing some insight into the redesign/maintain decision. Those tools which currently exist for the collection and measurement of performance data are highlighted. Suggestions are then made for yet other categories of data, difficult to collect and measure, yet ultimately necessary for the establishment of accurate predictions about the value of maintaining versus the value of redesigning.

Finally, it is concluded that this aspect of performance management deserves increased attention in order to establish better guidelines with which to aid management in making the necessary but difficult decision: maintain or redesign.

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

INTRODUCTION

There is no longer any doubt that some form(s) of structured specification, design, and implementation will be used to control major software projects of the eighties and beyond. The value to be gained from these approaches is well documented [1]. Unfortunately, many data processing departments will derive little benefit from these advances for years to come since most of the DP software budget is spent maintaining code that was not designed with these techniques.

How does management decide when to invest precious resources in the redesign of an existing, working application? Put another way, how do you quantify change that may result from cleaner, better implementation? Quantifiable costs associated with an application include: computer resources utilized by the application, programmer staff time plus computer resource costs expended to maintain the application, and associated user time spent trying to learn and to use the end product. Other costs which are harder to quantify include: net effects of frustration among users and maintainers of the existing application, possible increases in user productivity, overall improvements in operations resource consumption, and those expenditures associated with redesign and re-implementation.

Several semi-formal systems of rules have been developed with the goal of avoiding poor designs and implementations. Further, a type of folk wisdom has evolved that allows applications to be classified as generally good or bad, often based upon their resource consumption and/or the quality of the user interface. Tools exist [2, 3, 4, 5] for quantifying expected improvements in resource consumption resulting from specific design changes. While some recent studies have shown the value of the concept of software engineering, most design methods ignore performance, thus devaluating these tools. Furthermore, the actual maintenance costs are often much greater than the total computer costs associated with use of an application.

Less attention has been focused on quantifying just how costly an application is by utilizing as a base the human time invested in maintainence, an activity that consumes large fractions of a typical budget. It is necessary but insufficient to know how many man hours per month are invested in maintainence of an application. It is also necessary, but insufficient to understand why and how the maintainence was done. Other metrics are needed if good models are to be produced that allow a quantification of the benefits that might accrue from a complete redesign of the application.

In what follows, the performance metrics which were developed for evaluating an applications maintenance effort are described. The case study is a beginning towards providing management with decision making tools, but it is not totally adequate.

Presented here is the following:

- sample values of data collected for in-house maintenance effort,
- rewrite payoff estimates using only collected data,
- other, known approaches for collecting additional data which could improve predictions,
- suggestions for data which, given ideas for collection, would enhance the metrics necessary to parameterize models.

CASE STUDY

Maintenance work can be divided among three categories: fixing "bugs", adding minor enhancements or altering features, and developing new features for an existing system. The latter of these three categories may include, for example, a new output report which must rely on existing subroutines and file structures.

Described here in detail is the analysis of all maintenance work done for a large application that was ported (straight conversion) from a CDC 6600 to a Hewlett Packard 3000 mini-computer. The data was collected over a 13 month period. There are approximately 120 programs and subroutines written primarily in the Fortran language (with a few COBOL codes), consisting of approximately 43,000 lines of code. The application is used interactively by 80-100 users. Batch production cycle output is distributed to about 2,000 users. Nearly 140mb of disk storage is consumed for program source, binaries and data files. The basic function of the application is to provide a subset of users with a method of interactively entering budget forecast data and iterating that data, and subsequently to provide all users with hard-copy reports depicting the current status of forecasts versus actual costs to date.

The maintenance programming effort is subdivided according to three main categories: pure corrective maintenance ("fixes" to performance or implementation failures), enhancements or optimization (changes in the processing environment), and development or extension (addition of new features for increased performance). Table 1 reflects the number of person hours of effort expended for each of these three categories. While "development" effort is the largest of the categories, it should be noted that 85% of the development was expended on only two requests: documentation of the system and a series of new output reports. The largest number of changes fell into the "enhancement" category, as illustrated in Figure 1. Service Request numbers, the documentation form used to log changes, are not chronological in this figure, but represent grouping of effort. For example, in the curve labeled "E" for enhancement, the x-axis point labeled "2" indicates that one enhancement request required 150 hours of effort. On the enhancement curve, the x-axis points labeled "6", "7", "8" and "9" indicate that each of four changes required 50 hours of effort. Similarly, on the "development" curve, it is shown that only one request (the one for new output) required 1,000 hours. Figure 1 shows that the number of development changes was five, the number of enhancement changes was 40, and the number of maintenance

changes was 25, totaling 70 service requests. Figure 2 makes no distinction among development, enhancement or maintenance, but shows, as one curve, that the majority of changes were small in terms of effort hours. For example, 26 changes required 10 or fewer than 10 hours, 15 changes required 20 or fewer than 20 hours, etc. Compared to development, where one change took over 1,000 hours, maintenance and enhancement changes requiring less than 125 hours numbered 65. It can be inferred from this data that the "quick fix" or "patch" consumed over one-half of all effort expended during the time period. That is, enhancement and maintenance consumed 3,100 hours while development consumed 2,500 hours.

The two easily quantifiable costs associated with an application are computer resources for use and maintenance, and personnel resources for maintenance. Computer costs in this case study are fixed - approximately $5,900 per month for lease price plus approximately $1,100 per month for hardware maintenance, the total approximating $7,000 per month. The total number of person hours expended in maintaining the operation over the 13 month period is 5,639 - when overhead factors are added, the figure escalates to 7,933, or an effective average of 3.8 full time professional employees. Average cost per employee, including salary, overhead costs and fringe benefits exceeds $100,000 per year or $8,333 per month.

Total manpower cost is then calculated in Equation (1):

$$Tm = Ne * Na * A \qquad (1)$$

or

$$Tm = 3.8 * 13 * \$8,333$$

$$Tm = \$411,650$$

where Tm = total manpower cost
Ne = number of employees
Na = number of months elapsed
A = Average monthly cost per employee

Equation (2) gives total computer costs.

$$Tc = (Pl + Pm) * N \qquad (2)$$

or

$$Tc = (\$5,900 + \$1,100) * 13$$

$$Tc = \$91,000$$

where Tc = total computer cost
Pl = computer lease price
Pm = computer hardware maintenance price
N = number of months elapsed.

Comparison of computer costs of $91,000 to personnel costs of $411,650 points out that, in at least some applications, the ease of maintaining and using a system is more important than machine resource utilization.

Note that only programmer effort was included in these calculations. Management time, an even higher cost, was not considered, nor was production operations staff time. It is not known how many hours of effort were expended by users of the application, a metric which would be useful in the redesign decision. Based on informal observation, the user interface could be improved, but that kind of time-consuming survey has not yet been conducted in this case study.

Those who maintain the application complain that changes are difficult to make and test. All agree that a rewrite is necessary. Yet the application is "working", thus creating the need to prove to management that a redesign using modern methods would result in a smaller maintenance effort and improved overall costs. Quantifying a "payoff" to be gained from a rewrite is the key. The following three figures are an attempt to use the collected data to indicate payoff possibilities.

In strictly monetary terms, the current and anticipated expenditures of maintenance, both staff and hardware, can be plotted as a function of time. In figures 3, 4, and 5:

```
T1 = t * (C + P)
where T1 = total expenditures without rewrite
      t = elapse time, in months
      C = computer costs, defined as constant
          $7,000/month
      P = personnel costs
        = Ne * A
        = 3.8 * $8,333
        " $31,000
        where Ne = number of employees
              A = average monthly cost per person
```

Figure 3 is a realistic estimate, assuming that the application can be rewritten by three people in a nine-month period. Figure 4 is an optimistic estimate, assuming that the application can be rewritten by two people in a six-month period. Figure 5 is pessimistic, assuming it would take four people one year to accomplish a rewrite. Figures 3, 4, and 5 use the line T2 to indicate redesign costs. In all figures the following assumptions are standard:

1) computer resource costs will remain constant,
2) following a rewrite, maintenance programmer costs will be significantly reduced (it will take only two people to support the maintenance effort, instead of the current 3.8),
3) the maintenance, enhancement and development load (number of service requests) will remain constant.

Costs, given a rewrite is calculated:

```
T2 = t * (C + P/p) + R
where t = elapse time, in months
      C = computer costs, defined as constant
          $7,000/month
      P = personnel costs
        = Ne * A
        = 3.8 * $8,333
        = $31,000
        where Ne = number of employees
              A = average monthly cost per person
              p = reduction factor (assume 1.9)
              R = rewrite costs
                = Nr * A * t
                where Nr = number of employees
                           required for rewrite
                      A = average monthly cost per
                          person
                      t = elapse time, in months,
                          for rewrite.
```

In every case, costs are higher during the rewrite period when the project utilizes additional staff.

Also, in every case, costs plummet following completion of the rewrite, and the payoff period begins some time after the rewrite. Figures 3 & 4 show the savings effected - it can be seen that $120,000 and $245,000 are the savings by the 24th month for the realistic and optimistic cases, respectively. Payoff for the pessimistic case will not begin until sometime after the 12th month following rewrite. The total savings is the difference of T1 - T2. In figure 3 (realistic), the payoff begins six months after the rewrite, but in Figure 4 (optimistic) the payoff is shown to begin one short month after the rewrite.

These figures are a good indication of the positive effects of a rewrite, but they are incomplete without incorporating another needed metric - user interface. If user costs were added, T1 would shift upward, and T2 would shift upward, at least until the time of rewrite completion. If, however, there was some mechanism for predicting improved ease of use after rewrite (and therefore lower user costs), it could possibly be shown that T2 would dramatically shift downward after rewrite, creating greater overall savings.

Another metric needed to accurately forecast payoff possibilities is the cost of re-implementation. Guessing at the number of people and the time required for the project is not a good tool. The estimates used in this case study were deemed reasonable by all who were associated with the project, yet there is no guarantee that the application can be recreated and improved on a predetermined schedule.

OTHER APPROACHES

It has been said that "Maintenance-oriented design constraints are essential for continued reliability and correctness in a changing environment. The pressure for change results from the economic advantages of new hardware and the service advantages of new function. Since these advantages develop frequently, reflecting the rate of technological improvement and new applications, the pressure for change is persistent. Without the support of maintainability in such an environment, reliability and correctness are fragile properties which can be quickly lost to either error or obsolescence. Therefore, in a changing environment, maintenance-oriented design constraints become as important as those of presently-required function." [6]

Some directly measurable maintainability factors have been defined as:

- problem recognition time,
- administrative delay time,
- maintenance tools collection time,
- problem analysis time,
- change specification time,
- active correction time,
- local functional correctness test time,
- global correctness test time,
- independent change - audit time,
- recovery time or correction implementation time [7].

In the case study, the above ten factors were measured as one, although each of the ten could have been measured independently. What was not measured, due in part to lack of tools, were the potentially highest cost savers: possible increase in user productivity and an accurate figure for decreased maintenance cost after rewrite. Decreased maintenance costs were merely guesses, while user interface was left constant due to absence of data. For example, if the programs were redesigned with better user features, the enhancement load (number of service requests) could be expected to drop sharply.

While maintenance for the application is expected to improve following a rewrite, publication of an accurate expectation requires both additional data and additional measurement tools for projections.

Additional data might include:

- distribution of user time spent exercising each interactive feature,
- distribution of user time spent exercising each batch report feature,
- prioritization of the usefulness of the features by the users of the application (frequency of use is often not indicative of importance - many reports produced only at month end carry heavy significance),
- prioritization of future enhancements desired by users.

UNOBTAINABLE (?) DATA

Projections which, made accurately, would aid in the maintain/redesign decision might include:

- cost to redesign and re-implement,
- resource consumption of the rewritten version of the application,
- improvements in interactive user productivity,
- improvements in batch report user productivity,
- improvements in operations staff application use,
- improvements in production control.

CONCLUSION

Applications which are difficult to maintain or use are often considered candidates for redesign, even though they exist in working status. Many DP shops are handicapped by too few available methods for deciding when an application should be left in maintenance mode with known, fixed costs, and when it should be rewritten. Some advantages of a rewrite are:

- improved user features,
- improved maintainability due to structured specification, design and implementation,
- improved user and maintenance costs.

Redesigns are expensive as most applications must continue to serve the user with a maintenance staff during the rewrite period. Management requires tools for estimating the cost of redesign and plotting it against the potential cost savings of a new version.

Maintenance effort and computer resource consumption are easily quantified and were done so in the case study. Other metrics such as frequency of the use of an application feature by users and user satisfaction survey could be used in the study, yet they still would not provide enough information for an intelligent decision. Performance management aspects which need to be addressed in order to aid in the maintain/redesign decision are:

- how to accurately estimate programmer time associated with redesign,
- how to accurately predict resource consumption after rewrite if the end product remains the same, and if it does not,
- how to accurately estimate user productivity following rewrite,
- how to accurately estimate maintenance effort required for a new version.

Until each of these metrics are considered in the comparision of maintenance costs versus redesign costs, a proper decision cannot be clear.

TABLE #1

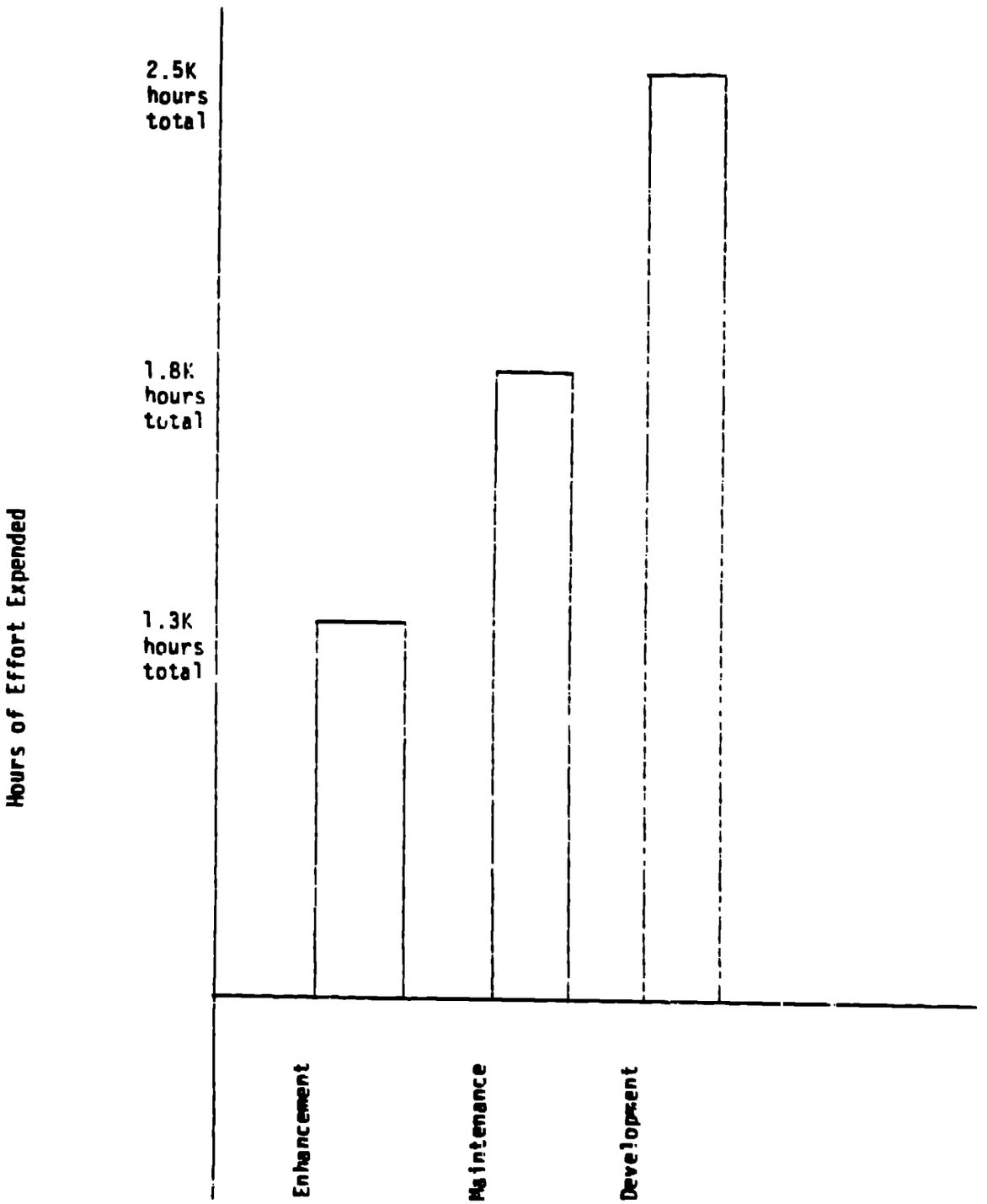

2.5K hours total
1.8K hours total
1.3K hours total
Hours of Effort Expended
Enhancement
Maintenance
Development
Categories of Effort

Figure 1

Person hours per change

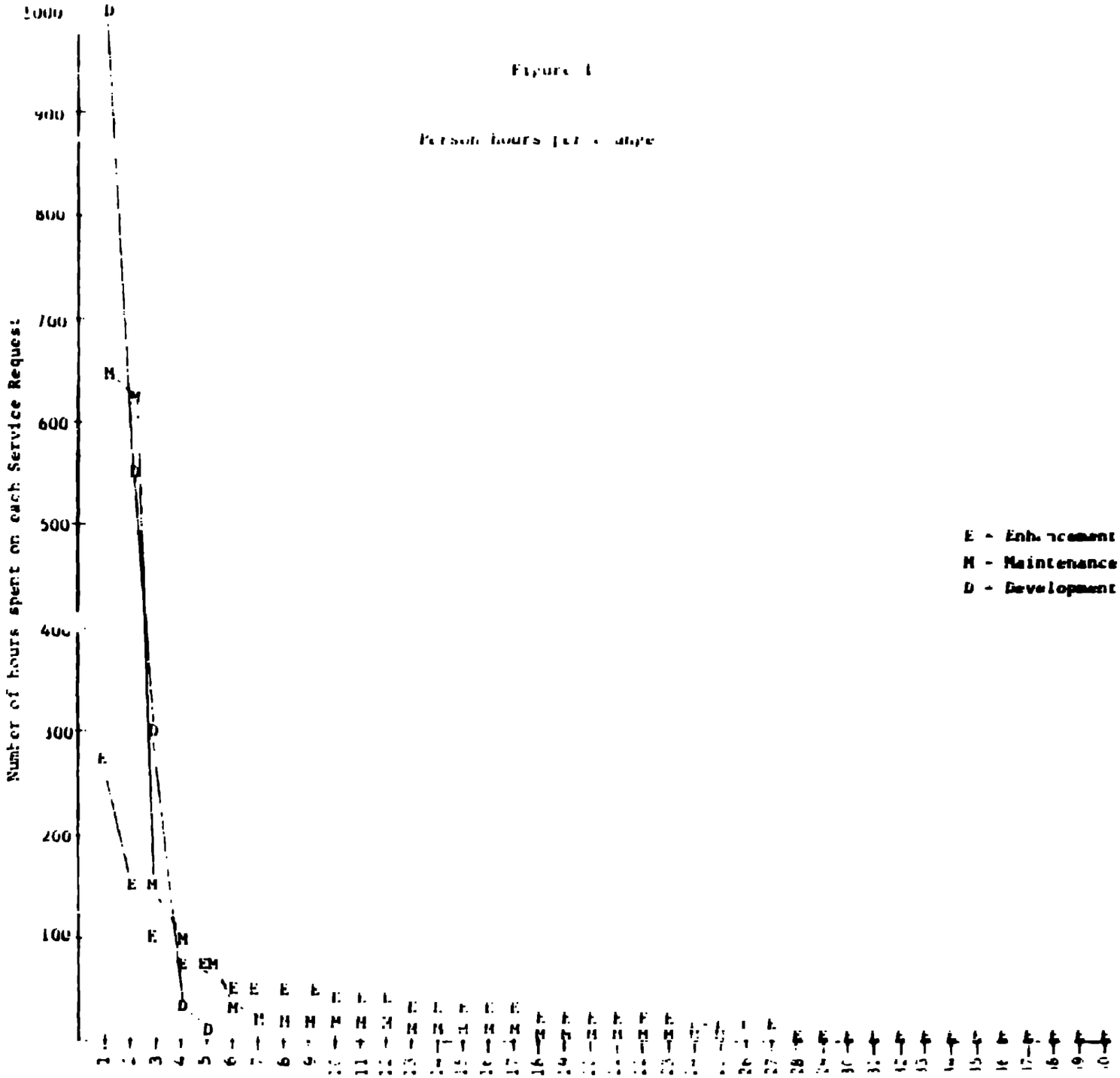


Figure 2

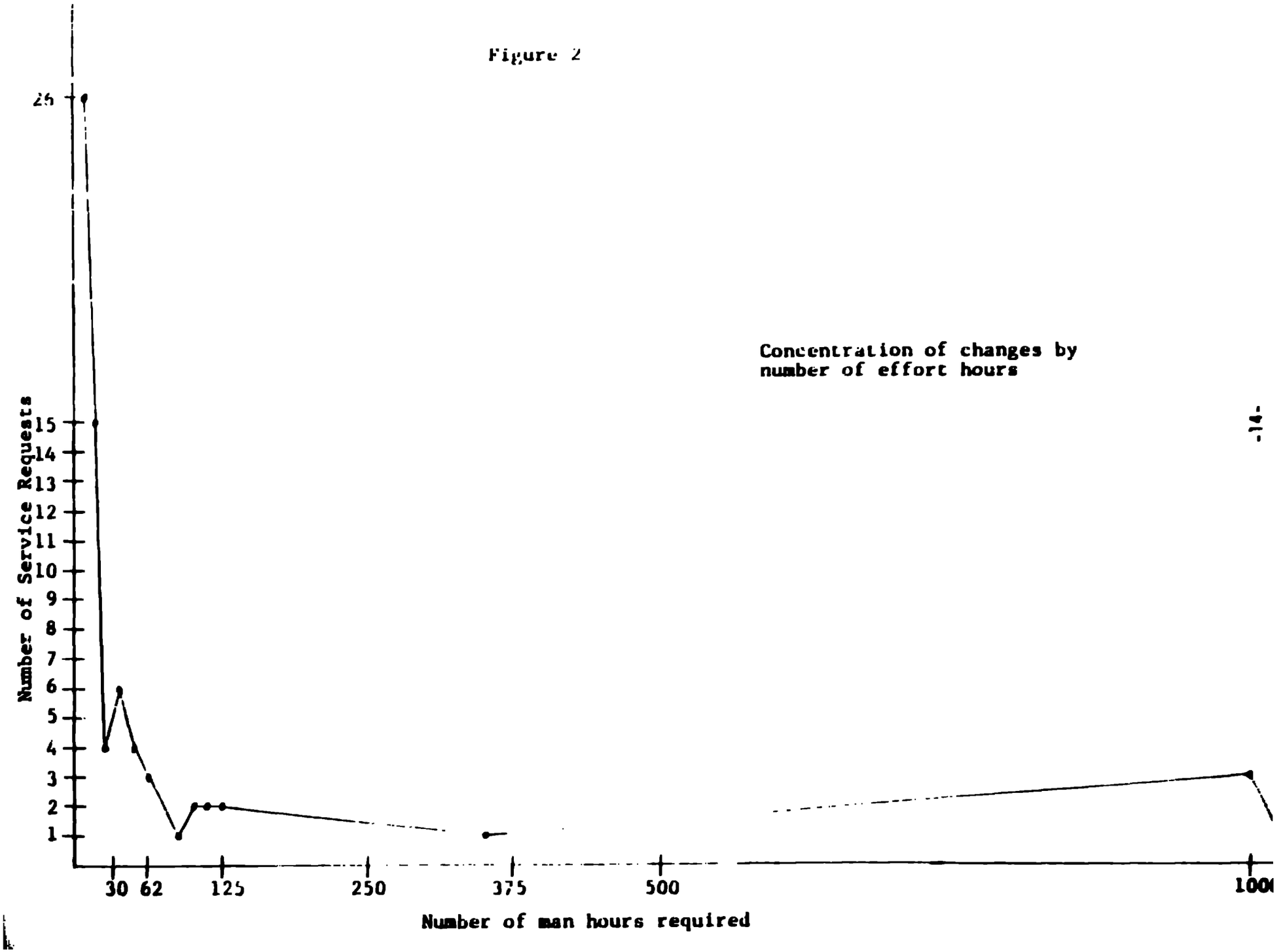
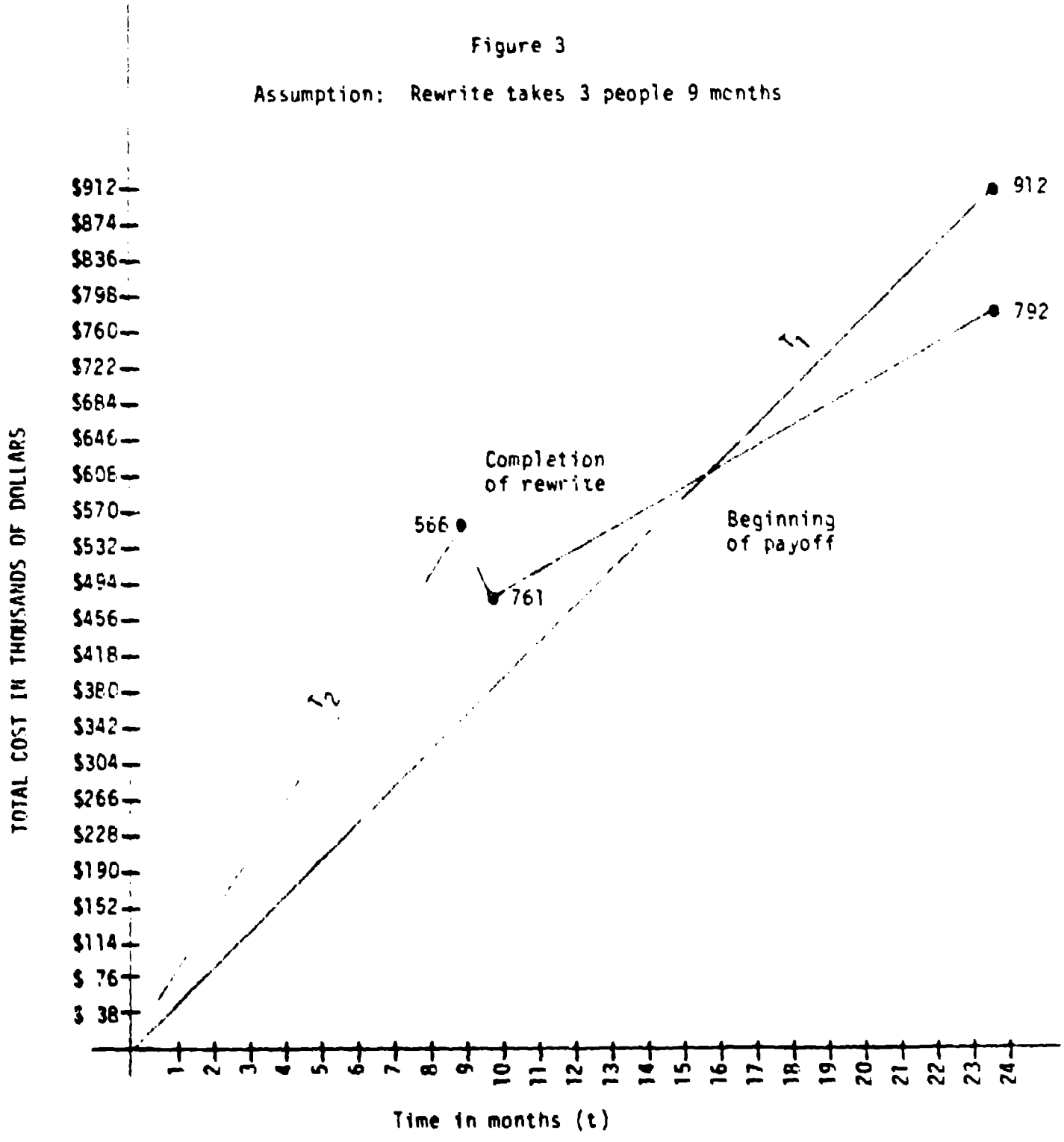

25
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
Number of Service Requests
30
62
125
250
375
500
1000
Number of man hours required
Concentration of changes by number of effort hours

Figure 3

Assumption: Rewrite takes 3 people 9 months

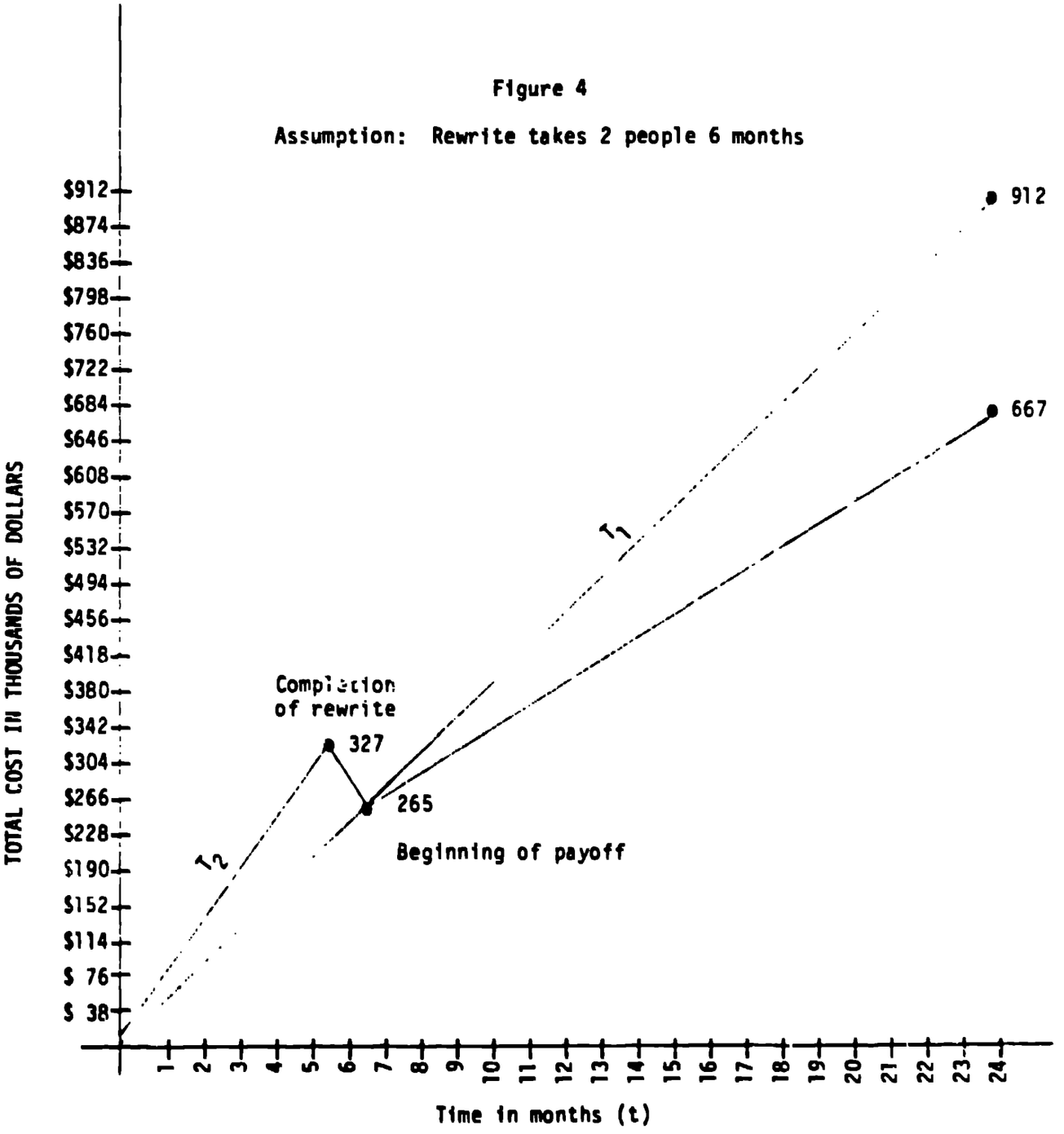


T_1 = Current cost of maintenance

T_2 = Rewrite cost plus maintenance cost

Figure 4

Assumption: Rewrite takes 2 people 6 months

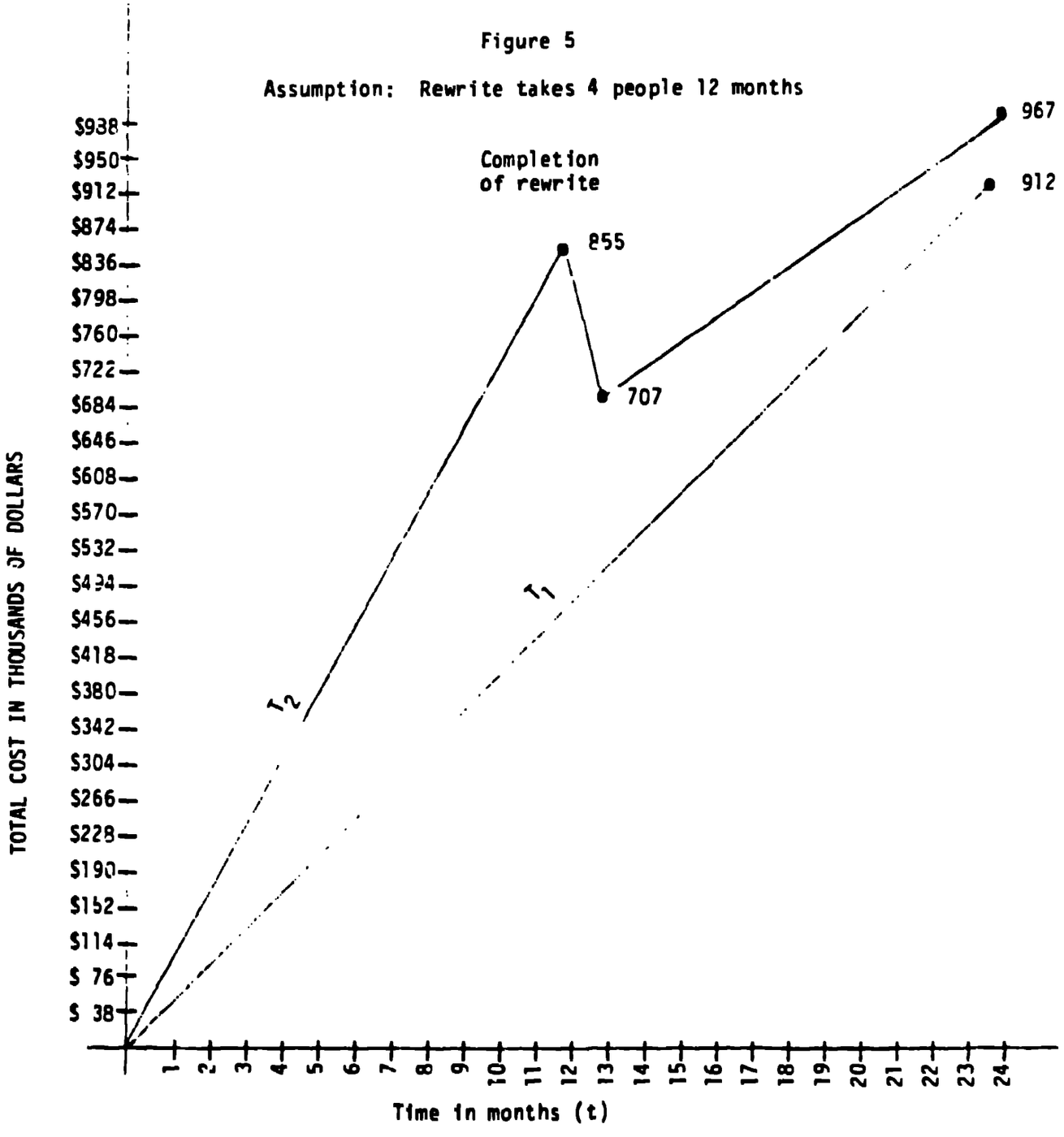


T_1 = Current cost of maintenance

T_2 = Rewrite cost plus maintenance cost

Figure 5

Assumption: Rewrite takes 4 people 12 months



T_1 = Current cost of maintenance

T_2 = Rewrite cost plus maintenance cost

1. Kleine, Karl, "Selected Annotated Bibliography on Software Engineering", ACM SIGSOFT, Software Engineering Notes Vol. 3, No. 1, January 1978.

2. C. U. Smith, J. C. Browne, "Aspects of Software Design Analysis: Concurrency and Blocking", Proc. Symposium on Computer Performance, Modeling, Measurement and Evaluation, Toronto, Ontario, Canada, May 1980.

3. C. U. Smith, J. C. Browne, "Performance Specifications and Analysis of Software Designs", Proc. Conference on Simulation, Measurement and Modeling of Computer Systems, Boulder, August 1979.

4. C. U. Smith, J. C. Browne, "Modeling Software Systems for Performance Predictions", Proc. Computer Measurement Group X, Dallas, December 1979.

5. T. J. Gilkey, F. R. White, and T. L. Booth, "Performance Analysis as a Software Design Tool", Proc. COMPSAC77, IEEE Computer Society, November 1977.

6. D. Gelperin, "Testing Maintainability,", ACM SIGSOFT, Software Engineering Notes, Vol. 4, No. 2, April 1979.

7. T. Gibb, "A Comment on 'The Definition of Maintainability'", ACM SIGSOFT, Software Engineering Notes, Vol. 4, No. 3, July 1979.